# AquaCore: A Programmable Architecture for Microfluidics

Ahmed M. Amin, Mithuna Thottethodi, T. N. Vijaykumar
Steven Wereley[†], and Stephen C. Jacobson[ϒ]

School of Electrical & Computer Engineering, Purdue University
[†]School of Mechanical Engineering, Purdue University
[ϒ]Department of Chemistry, Indiana University
{amamin, mithuna, vijay, wereley}@purdue.edu, jacobson@indiana.edu

## ABSTRACT

*Advances in microfluidic research has enabled lab-on-a-chip (LoC) technology to achieve miniaturization and integration of biological and chemical analyses to a single chip comprising channels, valves, mixers, heaters, separators, and sensors. These miniature instruments appear to offer the rare combination of faster, cheaper, and higher-precision analyses in comparison to conventional bench-scale methods. LoCs have been applied to diverse domains such as proteomics, genomics, biochemistry, virology, cell biology, and chemical synthesis. However, to date LoCs have been designed as application-specific chips which incurs significant design effort, turn-around time, and cost, and degrades designer and user productivity. To address these limitations, we envision a programmable LoC (PLoC) and propose a comprehensive fluidic instruction set, called AquaCore Instruction Set (AIS), and a fluidic microarchitecture, called AquaCore, to implement AIS. We present four key design aspects in which the AIS and AquaCore differ from their computer counterparts, and our design decisions made on the basis of the implications of these differences. We demonstrate the use of the PLoC in a range of domains by hand-compiling real-world microfluidic assays in AIS, and show a detailed breakdown of the execution times for the assays and an estimate of the chip area.*

**Categories and Subject Descriptors**
C.0 **[Computer System Organization]:** General - *Instruction set design, System architectures*
**General Terms:** Design
**Keywords:** Microfluidics, Programmable lab on a chip, Fluidic instruction set, Fluidic microarchitecture

## 1 INTRODUCTION

Microfluidics is a branch of engineering that deals with small quantities of fluids (of the order of nano or pico liters). Over the past fifteen years, microfluidics research has explored lab-on-a-chip (LoC) technology to enable miniaturization and integration of biological and chemical analyses to a single chip comprising channels, valves, mixers, heaters, separators, and sensors [16]. Interest in microfluidics has grown considerably primarily because these miniature instruments appear to offer the rare combination of faster, cheaper, and higher-precision analyses compared to conventional bench-scale methods. Using these devices, biologists and chemists have been able to manipulate molecules, fluids, and cells with precision and accuracy not achieved before. LoCs have been designed for applications in diverse domains such as proteomics, genomics, biochemistry, virology, chemical synthesis, and cell biology which are of interest to chemical, pharmaceutical, environmental and security monitoring, clinical diagnostics and food industries as well as to academic researchers in life sciences.

The LoC technology has moved well past the stage of proof-of-concept prototyping to being commercial products [1, 2]. However, to date, LoCs have been designed as application-specific labs-on-a-chip (ASLoC) where a new LoC is designed for every assay by creating and connecting on-chip components to match the steps of the *assay*, which is the detailed step-by-step specification of a particular analysis (i.e., assays are fluidic algorithms). This application-specific approach has two limitations. First, there is considerable design effort, turn-around time, and cost. Second, the tight coupling between the LoC and the assay reduces the productivity of both LoC users (e.g., biologists, chemists, and drug designers) and LoC engineers by requiring cross-disciplinary knowledge (i.e., the engineers must know the assay details and the users need to know the constraints of the specific LoC technology).

A programmable LoC (PLoC), which is a single design that can be programmed to run any assay, would address these limitations. The PLoC has the obvious advantages over ASLoCs of faster turn-around time, lower cost, and higher productivity for both LoC engineers and users. The current LoC technology (comprising valves, channels, mixers, and heaters), though being used to build ASLoCs, is mature enough to enable the PLoC's microarchitecture. Thus, the time is right for exploring the PLoC.

Previous work has explored programmability in a limited sense. Current et al. [8] automate droplet movement on a two-dimensional grid in a droplet-based LoC but do not provide programmability in the rest of the chip. Su et al. [23] propose a CAD approach for fast development of ASLoCs. Shaikh et al. [21] propose a breadboard approach which retains ASLoCs but allows modular LoC component designs that can be plugged together on a breadboard to form a complete LoC. Though this approach reduces the development time by reusing component designs, each LoC still requires days to develop, as reported in the paper. Recently, Urbanski et al. [26] replace these limited approaches with the pioneering idea of making LoCs fully programmable. In later work [24], they focus on a new programming language for microfluidics called *BioStream*, while we focus on the instruction set and microarchitecture.


*ISCA'07*, June 9-13, 2007, San Diego, California, USA.

Our first contribution is a comprehensive fluidic instruction set, called *AquaCore Instruction Set (AIS)* — pronounced as "ice" — and a fluidic microarchitecture, called *AquaCore*, to implement AIS (in [24], the authors describe only one operation — mix-and-store — and its implementation). We envision a compiler that will automatically translate high-level assays into AIS (though the details of the compiler is beyond the scope of this paper). Though fluidics and electronics are disparate technologies, there are similarities between AIS (and microarchitecture) and computer instruction sets (and microarchitectures) such as one operation per instruction and the separation of storage and functional units. However, AIS and AquaCore differ from their computer counterparts in four key aspects.

First, computers are universal (i.e., they can perform any computation) due to Turing-Completeness of instruction sets and their implementation. Unfortunately, fluidics lack such theoretical guarantees. Therefore, we must turn to empirical analysis of AIS's coverage of real assays from different domains. Our second contribution is such an analysis showing coverage, execution time, and chip area (in [24], the authors show mixing of fluids, but not any real assays).

Second, most computer instruction sets associate the input and output operands for the instructions with some storage location (register or memory). This practice is based on the facts that instruction outputs are used more than once, and that the uses are separated over at least a few instructions. Therefore, it is logical for a producer instruction to place a value in a storage location and for the consumer instructions to read the value from the location. In contrast, we observe that (1) in most fluidic assays, the output of a step is almost always used exactly once, (2) in certain fluidic technologies fluid movement uses valves which, unlike electronic transistors, may fail after just a few thousands of actuations (e.g., flow-based technology that moves fluids using peristaltic action), and (3) in those technologies moving a fluid is as slow as performing an operation. Combining these three observations suggests that it is not only unnecessary but actually detrimental to associate fluidic operands with storage locations which would require moving the fluids from storage to functional units and back. Instead, we avoid this association and propose storage-less operands which are moved to and from functional units using explicit move instructions. These explicit moves allow the output of a functional unit to be transferred directly to the input of another unit without involving any storage location. One could use conventional instructions and then optimize away unnecessary storage locations, but our approach is simpler.

Third, volumes of fluidic operands are different from sizes of computational operands which are a few fixed-size datatypes supported by the hardware (e.g., 8, 16, 32, and 64 bits). Smaller operands are handled by padding available datatypes, and larger ones by composing available datatypes. However, fluidic operands are of many different volumes which, unlike computational operands, cannot be padded easily to a fixed larger volume (by topping off) in cases where reagent concentrations have to be maintained. We make the key observations that the need for padding fundamentally comes from using fixed-size datatypes; and that unlike computational hardware, fluidics hardware can naturally operate on virtually any volume up to resource capacity without needing any topping off (e.g., mixers can mix fluids of virtually any volume up to the size of the mixer). We exploit this natural flexibility by allowing variable-volume operands for AIS instructions. Because high-level assays specify operand volumes only in a few steps while the rest of the steps use implicit volumes, most AIS instructions operate on implicit, yet variable, volumes which are not specified. Further, high-level assays typically use relative volumes (e.g., mix one part of reagent *A* with two parts of reagent *B* without specifying the absolute volume for a part), and forcing the use of absolute volumes would encumber programming. Consequently, AIS allows relative-volume operands. We rely on the run-time system to translate the relative volumes to implementation-specific volumes while maintaining portability across same-technology implementations. However, we must accommodate assay-steps that handle absolute volumes of fluids. To that end, AIS allows absolute-volume operands for some instructions.

Finally, while computational hardware is homogenous in that both control and datapath use the same VLSI technology, LoCs are heterogeneous in that they traditionally employ electronics for control and fluidics for fluidpath (analogous to datapath). Though control logic could be implemented using fluidic hardware (as alluded to in [11]), it is best to implement control logic in electronic hardware for reasons of speed, reliability, and technology maturity. Of course, the fluidpath has to use fluidic hardware. Recent work to automate droplet movement [23] also advocates this heterogeneous approach, as does BioStream. Accordingly, AquaCore is also heterogeneous.

Given the infancy of programmable fluidics, we view AIS and AquaCore as a first step in the design of instruction sets and microarchitectures for programmable microfluidics. We expect that our ideas will be improved upon in the future, much like current technologies which have matured through several successive refinements.

To summarize, our key contributions are:

- we propose AIS in which (1) operands are not always associated with storage, and (2) instructions operate on implicit variable volumes;
- we propose AquaCore, a heterogeneous PLoC microarchitecture, which uses electronic control logic and fluidic datapath; and
- we show that AIS can cover real-world assays from many domains. In addition, we also show a detailed breakdown of the execution times for the assays and an estimate of the chip area.

The rest of the paper is organized as follows. Section 2 provides background. Section 3 describes AIS and AquaCore. Section 4 shows our results using real-world assays, and Section 5 concludes.

## 2 BACKGROUND AND RELATED WORK

Over the past fifteen years, a number of academic and commercial groups have been exploring both fundamental issues and applications of microfluidic devices for chemical and biochemical analysis (i.e., lab-on-a-chip technologies) [1, 2, 20, 3, 27]. Lab-on-a-chip solutions have been demonstrated for diverse applications including detection of end-stage renal disease [7], detection of point mutation in specific genes [1], antibody analysis, protein purification and polymerase chain reaction (a commonly used technique to amplify DNA) [1]. These microfluidic devices are typically fabricated using micromachining techniques including

photolithography, etching, thin film deposition, and coverplate bonding. Compared to state-of-the-art microelectronics, these devices are relatively simple in design and moderately easy to fabricate. Progress continues to be made in a number of areas and these advances have been recently reviewed [9]. However, as one looks through these examples, one observes nearly as many chip designs as there are assays. This trend confirms the compelling need for programmable labs-on-a-chip (PLoC).

There are two dominant technologies for microfluidics — flow-based [25] and droplet-based [19]. While flow-based LoCs operate on fluids that flow either continuously or as discretized volumes, droplet-based LoCs manipulate liquid droplets. In the remainder of this section, we focus on flow-based LoCs though droplet-based equivalents exist. We demonstrate that the enabling technology for the various fluid-path components that are needed for PLoC exists or is being created (e.g., [14, 12, 11]), and the time is right for exploring programmability. Broadly, the components needed for the fluid-path of a PLoC are (a) reservoirs to serve as temporary fluid storage, (b) fluid functional units (FFUs) to execute various primitive operations such as mixing, heating, separating, and sensing and (c) a programmable connectivity fabric to tie all the components together and to route fluids among the reservoirs and FFUs as required. Each of these components are described below.

**Reservoirs**: Fluid reservoirs are basic components with support to dispense volumes of fluids and are relatively simple to implement [16].

**Mixing**: A large variety of microfluidic mixers have been demonstrated previously. These range in sophistication from simple, passive structures that permit routing of two or more fluids into reaction chambers/ channels in which mixing occurs by diffusion or simply by flowing together to more sophisticated, *active* mixers that include micro-electro-mechanical agitator mechanisms or pumps to accelerate mixing [16].

**Heating**: Thermal cycling, heating and incubation at specified temperatures are commonly used in many assays. Several alternative implementations including resistive heaters, thin-film heaters and ceramic heaters have been demonstrated [16].

**Separation**: Microfluidic separation devices that separate fluids into components based on affinity, size/ weight, charge or hydrophobicity have shown performance either equivalent to or better than conventional laboratory devices, and appear to offer the rare combination of better-faster-cheaper simultaneously [16].

**Sensing**: The outcome of an assay is often detected by measuring/sensing a physical [15] or electrochemical property of a fluid [16]. This requirement represents a challenge because (1) the number of sensor options is large and (2) some sensors are not miniaturizable. For example, some assays may need to detect the optical density of a mixture at certain wavelengths. Others may require measurement of fluorescence or may use mass spectrometry. While some sensors have been integrated on chip, others remain off chip. Off-chip sensing can be done via "windows" that expose fluids to external sensors. It is also possible to deploy electro-spray dispensers (similar to those used in inkjet printers) that spray fluid into external sensors.

**Valves, Pumps and Channels**: The key technology that enables programmability in LoCs is the valving. Valves and channels serve the same purpose in microfluidics that tristate-buffers and buses serve in digital computers. Valves allow dynamically changing the connectivity among the components of the microarchitecture as required by different instructions under execution. Microvalves have been known for a decade [10, 18] but have not been practical in highly-integrated LoCs due to complex fabrication procedures and chemical and physical incompatibilities.

A recently-developed latching valve [11] is easier to manufacture. We focus on this valve. The valves open or close depending on where there is vacuum or pressure at the control input provided by off-chip solenoid pumps. A latching valve maintains its open or closed state for a long time (a minute) even when disconnected from its pump. However, one issue continues to be a problem: the valves may fail after just a few thousands of actuations [11].

The valves can be arranged to function as peristaltic pumps (different from solenoid pumps) along a channel. The idea is to place three inter-locking valves along the channel. Opening a valve creates a vacuum in the valve's chamber and closing creates pressure. The valves are opened and closed in a sequence such that vacuum and pressure are created at the right valves, moving the fluid in the required direction.

Instead of using peristalsis, droplet-based LoCs move the droplets under an electric field by collapsing the surface tension of the droplets causing the droplet to tumble (this effect is called electrowetting [19]). To prevent droplets from evaporating, these LoCs coat the droplets with oil. However, some chemicals dissolve in oil making droplet-based LoCs unsuitable for assays using such chemicals. Also, separation is hard with droplets.

Finally, I/O interfacing to the chips has also been demonstrated in [17].

# 3 AIS AND AQUACORE

Principles of instruction set design and microarchitecture design for traditional computers are well understood. In this section, we focus on the key differences between the fluidic domain and the digital domain and the consequent design decisions in AIS and AquaCore microarchitecture. Recall from Section 1 that the differences are: lack of Turing-Completeness, storage-less-operands, nature of operand size, and heterogeneity of the microarchitecture.

## 3.1 Turing-Completeness

The theory of computation identifies the finite set of operations that computer instruction sets must support to be universal to perform any computation (i.e., Turing-Completeness[1]). Unfortunately, because fluidics lacks such theoretical guarantees, all-purpose PLoCs capable of executing any assay are not feasible. While basic fluidic operations of mixing and heating needed in most assays can be supported in all fluidic ISAs and microarchitectures, separation and sensing pose a serious challenge because of their diversity. For instance, a PLoC that senses only optical density (or only molecular weight) cannot sense chemical properties (or electrical charge) and therefore, cannot execute assays requiring chemical sensing (or electrical-charge separation). Supporting all known forms of separation and sensing is impractical (and would still not guarantee universality). Consequently, fluidic ISA and PLoC microarchitecture design must turn to empirical analysis

1. To be truly Turing-Complete, computers must have infinite memory but this requirement is not an issue in practice.

of coverage of assays from diverse domains. As such, AIS is designed to cover many domains.

## 3.2 Storage-less Operands

We make three observations that are unique to fluidic assays and technology that directly drive AIS's design. First, the basic rationale for the traditional dyadic instruction set in digital computers which operates on two source register (or memory) operands and produces a third register (or memory) value is that such values are often used more than once [5]. Thus, it is logical to save the value in a register (or memory location) that can be read multiple times. In sharp contrast, we observe that in a diverse set of assays intermediate fluids are typically used exactly once. In the few cases that a fluid is used multiple times, there is usually an explicit step in the assay that divides the fluid into multiple parts, each of which has one subsequent use. Such explicit division is not surprising given that uses are destructive in fluidics (while uses are non-destructive in digital computers) and thus fluidic-assay writers create explicit divisions if they expect multiple uses.

The above "single-use" observation eliminates the need for associating intermediate results to storage locations (reservoirs) after each operation. One may argue that if there were no penalty to moving intermediate fluids to storage locations (even if there are not multiple uses of the fluids), then the single-use observation does not really have a direct bearing on the instruction set design. However, our second observation is that in certain fluidic technologies fluid movement uses valves which, unlike electronic transistors, may fail after just a few thousands of actuations. Further, our third observation is that these technologies impose a high latency cost to move fluid between reservoirs and FFUs. These latter two observations eliminate the above no-penalty argument. For example, in peristalsis-based fluid transfer technology like the one we use in AquaCore, transfers require valve actuations and may be as slow as the operation at the FFU. Consequently, we propose storage-less operands which are not associated with any storage location. Instead, input operands are moved to, and output operands moved from, functional units using explicit move instructions. These explicit moves allow the output of a functional unit to be transferred directly to the input of another unit avoiding unnecessary operand movement between reservoirs to FFUs. Note that in droplet-based technology fluid transfer does not use failure-prone valves (Section 2) and is significantly faster than FFU operation. However, we design AIS for the failure-prone and slower technology which represents the tighter set of constraints. By using explicit moves in droplet-based technology, we may unnecessarily incur some code bloat. Fortunately, as we explain in Section 3.4, AIS code is handled only by electronic hardware and not by fluidic hardware, and electronic memory is large enough to absorb any such bloat.

Table 1 lists the AIS instructions which may be classified into two broad categories. First, the *move* instruction is in a category of its own as a fluid transfer instruction which requires an explicitly named source FFU, reservoir, or port and a destination FFU, reservoir, or port. Note that all FFUs, reservoirs and ports are identified by a unique identifier in a common identifier-space. We will discuss the optional volume field in Section 3.3. Second, all other FFU-based instructions (such as *mix*, *heat*, *sense* and *separate*) form a second class of instructions with storage-less operands. These instructions implicitly operate on operands that have been previously loaded into the FFU by the *move* instruction. The output fluids of FFUs are similarly held in the FFUs until a *move* instruction explicitly removes the fluid. AIS exposes transfers to the compiler via the *move* instruction. It may be possible to use traditional dyadic instructions and then dynamically eliminate unnecessary transfers to reservoirs by directly moving the fluid from one FFU to the next. However, our approach is simpler and easier to implement. It may appear that a dyadic instruction set which can specify arbitrary identifiers as operands can also eliminate unnecessary transfers by directly sending intermediate fluids to the FFUs where the fluids are used. However, the fact that the subsequent instruction has no input operands makes it hard to compose instructions effectively.

Another challenge unique to fluidic instructions is detecting the completion of some fluidic instructions (e.g., *mix*) because instruction completion may depend on the machine parameters *and* the specific reagents being mixed. At first glance, this dependence may appear analogous to that in digital computers. For example, the delay through an adder depends on the input data as well as on the implementation. However, the variance in fluidic instruction completion times can be significantly larger than that in digital instruction completion times. Therefore, the solution used in digital computers of assuming the worst-case delay is not appropriate in the fluidic domain. The alternative of detecting mix completion at runtime is infeasible because hardware sensors to detect mix-completion for arbitrary fluids do not exist. AIS requires an explicit, assay-writer-specified completion time for mix and certain other instructions, as shown in Table 1. Tying completion times to instructions may appear similar to the idea of delay slots. However, delay slots taint architectural specification with implementation details for performance improvement whereas our completion times are needed for feasibility of implementation and not performance. Adding the time argument will not jeopardize the portability of AIS as long as there are no drastic differences in the functional unit implementations.

Finally, our storage-less operands have another advantage in that mixing of multiple (say $n$) ingredients incurs only a single mix-latency (and $n$ move latencies). A traditional dyadic microarchitecture would incur ($n$-1) mix latencies to mix $n$ fluids as it has to mix two fluids at a time. It may be possible to parallelize the mixes by using a combining tree, but that would require multiple mixers whereas we require only one.

## 3.3 Operand Volumes

Fluidic instructions operate on arbitrary volumes which results in several differences from traditional computer design techniques. Below, we discuss three key issues that arise due to this constraint and our techniques to address these issues.

First, the requirement for truly arbitrary volumes in the continuous sense is infeasible to implement because engineering constraints impose natural lower bounds on the volumes that fluidic hardware can handle. Instead, we assume that volumes are multiples of a small "least-count" unit which limits the resolution with which volumes can be manipulated, similar to limited floating-point precision. Our design uses a least-count volume of 1 nl compared to reservoir capacity of 100 nl, which gives sufficient resolution for all the assays we have seen. Note that even bench-scale

**Table 1: AquaCore Instruction Set (AIS)**

| Fluid Instruction | Semantics |
|---|---|
| **move** *id2*, *id1*, <*vol*> (*id2* is destination) | Move the contents of *id1* to *id2*. *id1* and *id2* can be a reservoir, fluid functional unit (FFU), input port, or output port. *vol* is an optional parameter to specify the relative volume to be moved. If no volume is specified, a default is used. Some combinations are illegal (e.g., move to an input port). |
| **move-abs** *id2*, *id1*, *vol* | Similar to **move** except *vol* is not optional and it specifies absolute volume in nl. |
| **mix** *id*, *time* | Mix the contents present in the mixer *id* for a duration of *time* seconds (*id* should map to a mixer FFU; if not, it is an error). Because mixing takes different times depending the reagents being mixed, the assay writer must explicitly specify the time to completion. |
| **separate.SIZE** *id*, *time* | Separate the contents of the size-based separator *id* into two parts based on their size by passing the contents through an inbuilt filter for a duration of *time* seconds. The separation unit works like a sieve and the two parts are stored internally in separate pools with unique ids that can be moved out later. |
| **separate.CE** *id*, $E_{sep}$, *len, time* | Using capillary electrophoresis (CE), separate the contents of the separator *id* into a continuous range of molecules that differ in mobility by waiting for a duration of *time* seconds. The contents pass through a CE column of length *len* under an electric field of $E_{sep}$. If *len* is more than the hardware length, the fluid is automatically cycled for multiple iterations. Separation medium (called buffer) is moved to the column's buffer port which has a unique id. For gel-based separation, the gel is moved and preloaded into the column's gel port (or it may be mixed with the buffer off-chip). CE separation is typically coupled with continuous sensing. (See **sense.FL**) |
| **separate.AF** *id*, *time* | Separate the contents of the affinity-based separator *id* into two parts — one that has affinity to the "affinity compound" and the other that does not — by waiting for a duration of *time* seconds. The two parts are held in separate pools with unique ids that can be moved out later. The FFU has a special internal pool (with a unique id) to hold the affinity-compound which is moved there. |
| **incubate** *id*, *temp*, *time* | Heat the contents in the heater *id* to a temperature *temp*, incubate the contents at that temperature for a duration of *time* seconds. Temperature is detected by a sensor built into the heater. |
| **input** *id2*, *id1* (*id2* is destination) | Input the fluid from the input-port *id1* into the reservoir *id2* (*id1* is assumed to be connected to the appropriate fluid), enough to fill the reservoir *id2*. This instruction is a pseudo-op and is translated into a **move**. |
| **output** *id2*, *id1* (*id2* is destination) | Output the fluid stored in the reservoir *id1* to the output port *id2* (*id2* is assumed to be connected to an off-chip reservoir or disposal). This instruction is a pseudo-op and is translated into a **move**. |
| **coated-plate-setup** *id* | Prepare the reaction plate *id* that needs to be coated with specific chemicals/biomolecules. The coating is done by pressure-evaporating the coating compound onto the plate. The compound is dispensed from an input port dedicated to the plate *id* and can be changed during execution by the user. **Coated-plate-cleanup** is similar. |
| **sense.OD** *id*, *senseval* | Sense the optical density of the fluid in the optical sensor *id* and copy the sensed value to the dry variable *senseval*. Instruction completion is detected by feedback from the sensor which is typically off-chip. |
| **sense.FL** *id*, *senseval[]* | Continuously sense the fluorescence (either native or that of added tags) in the contents of the CE column associated with the fluorescence sensor *id* and periodically copy the sensed value to the dry variable array *senseval[]*. This instruction completes when the corresponding **separate.CE** completes. |
| **concentrate.EV** *id*, *temp*, *time* | Assuming that the heater *id* contains a mixture of an analyte and a carrier fluid, concentrate the mixture by evaporating the carrier fluid by holding the mixture at temperature *temp* for duration *time*. Ideally, *temp* must be at or above the boiling point of the carrier fluid and *time* must be selected to achieve the desired concentration. Other methods for concentrating (e.g., electrophoresis, dielectrophoresis, and precipitation) may be adopted by extending AIS to add other variants of the concentrate instruction. |
| **dispose** *id* | Dispose of the contents of *id* which can be a reservoir or an FFU. This instruction is a pseudo-op and is translated into **move** *id*, *output-port-for-disposal.* |

assays often use stepper-motor-based dispensing which imposes a similar least-count constraint.

Second, discretization in terms of multiples of the least count does not eliminate the requirement for manipulating variable volumes of fluid which has no equivalent in computer design. Variable volumes is not analogous to operating on different computer data types because in the computer domain it is trivial to pad the smaller representation to the bigger representation by zero- or sign- extension. In fluidics, naively "padding" volumes to operate on equal volumes would fundamentally alter mixture ratios, violating the assay requirements. Biostream [24] offers an alternative method to achieve arbitrary mix ratios by performing a series of mix-and-store operations on a fixed unit volume. A single mix gives a ratio of 1:1. By using repetitive mixes of the resultant mixture to perform essentially a binary search over concentration ratios, they achieve arbitrary mix ratios.

Because fluidic hardware can naturally operate on variable volumes, we adopt the simpler method of allowing instructions to

operate on variable volumes. In doing so, we wish to avoid requiring the assay writer to specify volumes at *every* step because high-level assays typically specify volumes only at a few steps. To that end, we specify volumes only for *move*. All other instructions operate on implicit volumes: *input* moves as much fluid as needed to fill the reservoirs and all FFU instructions operate on the pre-loaded volumes.

Because some assays specify absolute volumes and some specify relative volumes, AIS supports two corresponding flavors of move (*move-abs* and *move*). An assay that uses absolute volumes will not be portable across implementations because FFUs/reservoirs in an alternate implementation may be too small to hold this volume. But by using absolute volumes the assay writer has precluded portability. In contrast, many assays are specified using relative volumes. For example, the assay may require mixing one part of reagent *A* with two parts of reagent *B* without specifying the volume of each "part". AIS includes the *move* instruction with support for relative volumes. Unlike *move-abs, move* is expected to be portable across same-technology implementations. To ensure such portability, we propose to translate the relative volumes to implementation-specific volumes at runtime. The translation is complicated by one key issue. An assay step may partition a fluid into $n$ portions for later use. However, if the fluid is naively partitioned into *equal* portions, the usage ratios of some portions may cause underflow. Consider an example, where fluid *A* is partitioned into four portions and one of the portions is to be mixed with fluid *B* in the ratio 10:1 (*A*:*B*). Then if *A*'s portion is small to start with then *B* may underflow (i.e., the volume needed may be smaller than the least count). If the partition step allocates *unequal* portions of *A* so that the portion that gets mixed with *B* is larger than the other portions, the underflow can be avoided. Note that such allocation can only minimize, but not eliminate, the possibility of underflow. It must be recognized that underflow in PLoCs, just like floating-point underflow in digital computers, is not a problem that can be prevented by design. Assays that underflow must be rewritten to avoid underflow.

In general, there are two fundamental ways to address this problem by ensuring that adequate volumes of all required reagents are available. Either the mechanism must look ahead in the assay to anticipate the required volumes and conservatively use fluids such that it lasts for the entire assay; or there must exist mechanisms to regenerate needed fluids if they are exhausted. BioStream takes the regenerate-on-demand approach. While elegant in theory, we believe that regenerating unanticipated fluids may place a high demand on LoC resources requiring unbounded LoC resources (or methods to mimic unbounded LoC resources by virtualization). While such virtualization is feasible for digital computers, microfluidic technology is not yet at that level of maturity. As such, we recommend the alternate approach of looking ahead in the program and conservatively expending fluid using unequal portions to prevent underflow, as mentioned above. As we discuss next, AquaCore' heterogeneity may enable this look-ahead with little performance penalty. We give a simple example of the look-ahead in Section 4.1 and leave the details for future work.

## 3.4 AquaCore Microarchitecture

Similar to traditional LoCs, the AquaCore microarchitecture is heterogeneous with two distinct parts. (1) The wet part/chip which operates on fluids (the fluid-path) and (2) the dry part/chip which acts as the control logic by interpreting AIS instructions and actuating the appropriate valves, pumps, and FFU controls. BioStream alludes to the possibility of exploiting this heterogeneity by observing that digital logic is faster by several orders of magnitude than microfluidic components and therefore could be used to perform runtime optimizations (e.g., look-ahead volume management). We leave other uses including the possibility of whole-program analysis and optimizations for future work.

AquaCore is illustrated in Figure 1. It shows the clean separation between the dry and wet parts. Because the wet and dry parts are different chips, there are no cross-fabrication issues. The main interaction between the wet and dry part is that the all-electronic dry part controls off-chip electrical solenoid pumps that control the valves in the wet part. The only issue is that the wet part may have hundreds of valves (as we estimate in Section 4.3) and the solenoid pumps are too large to allow as many pumps as there are valves. Fortunately, we can get away with much fewer pumps than valves.

Recall from Section 2 that the latching valves we use stay latched for a long time (minutes). This property enables a large number of independent valves to time-share a single controlling solenoid pump using time-division multiplexing [11]. The pump cycles through the valves under its control via a *fluidic* 1-to-n demultiplexer whose select inputs are also provided by solenoid pumps. To control $n$ valves, only $\log_2 n + 1$ pumps are needed [11]. The fluidic demultiplexer uses non-latching valves and is identical to a CMOS demultiplexer using pass gates. The fluidic demultiplexer takes about 120 ms to set one latch, and assuming each latch holds for 2 minutes, about 1000 valves can be set before the first latch needs refreshing [11]. Our microarchitecture needs about 100 valves which is within this limit. If more valves are needed then each group of 1000 valves would need a set of pumps and a demultiplexer.

We focus on the wet part as the dry part consists of well-understood digital logic. The wet part is organized similar to a three-bus microarchitecture. The three central fluid channels (A, B and C) are the bus-equivalents, and they transfer fluids from reservoirs or FFUs to other reservoirs or FFUs. Recall from Section 2 that the channels use latching valves to achieve peristaltic pumping. All the individual FFUs and the reservoirs are connected to the above channels via valves. The solid black ovals represent controllable valves that can be open or closed. Fluid movement is achieved by (1) opening and closing the appropriate valves, and (2) by ensuring fluid flow in the required direction by peristaltic pumping. We omit the peristaltic pumps necessary to move fluids from the figure for simplicity. The fluidpath also includes FFUs for mixing, heating, sensing, separating, and disposal. All these individual functionalities have been demonstrated in the context of ASLoCs (Section 2).

Note that unlike digital registers which can be "spilled" to memory if the number of registers are inadequate, current fluidic hardware has no equivalent off-chip fluid storage to handle assays that require more reservoirs than available. As such, having an adequate number of reservoirs is critical for correctness and not just for performance. Should off-chip storage become feasible, register spilling can be employed here as well.

A key concern for AquaCore operation is the reliability of the valves, as mentioned in Section 2. In contrast to digital computers which are reliable for years, currently implementable valves can

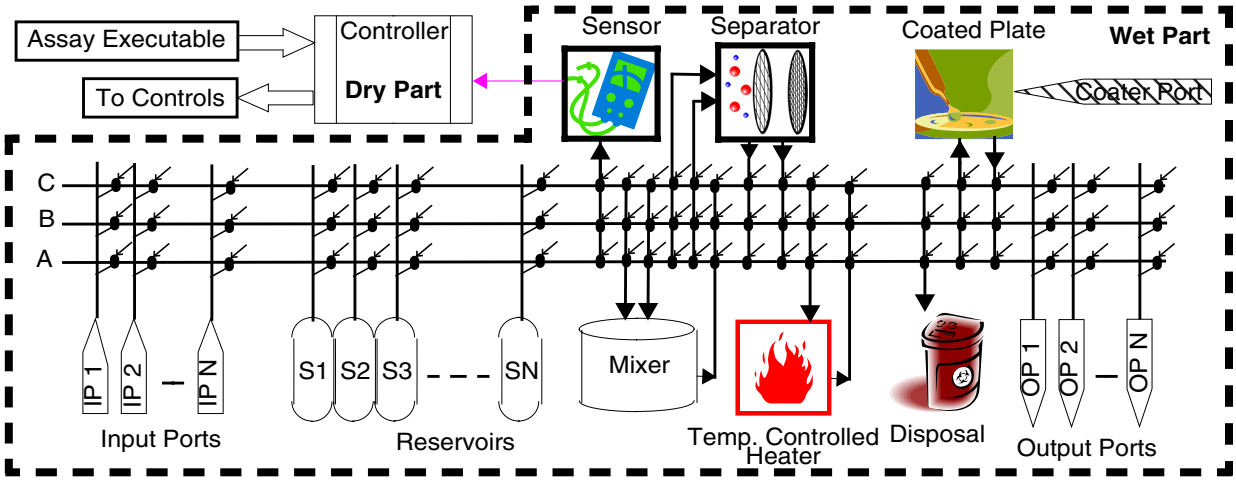


**FIGURE 1: AquaCore microarchitecture.**

operate for a few thousands of actuations [11]. To that end, the three channels in the microarchitecture share the burden of fluid transfers, reducing the number of actuations of each of the valves in the microarchitecture. However, adding more channels increases the chip area, creating a trade-off between area and reliability. Nevertheless, it remains to be seen if this issue will finally be resolved by (a) development of more reliable valves or (b) the development of an alternate business model (compared to digital computers) in which the disposable wet part of AquaCore will be discarded after failure.

Another concern is fluid contamination due to reuse of functional units or channels, either within steps in the same assay or between different assays. By adding wash/purge steps after every move (or FFU operation), this problem may be reduced for most assays,. However, such wash steps may be insufficient for some sensitive assays (e.g., clinical applications), in which case the chip should be discarded after one use.

Though our description has implied fluid flow-based implementation, a similar architecture would work for a droplet-based implementation (e.g., [19]) where droplets move between reservoirs and FFUs. However, AIS binaries may not be portable across such implementations due to the differences in their technologies (e.g., fabrication materials such as glass versus plastic).

# 4 RESULTS

As noted in Section 1, the key differences between a programmable architecture for fluidics and conventional computer architectures are: (1) the lack of a Turing-complete fluidic architecture which implies we need to evaluate the coverage by AIS and AquaCore microarchitecture of assays from disparate domains, (2) the single-use nature of intermediate results combined with failure-prone and slow movement of fluids which led us to use storage-less operands, (3) the nature of fluid operands which led us to use variable-, implicit- and relative-volume operands, and (4) the heterogeneous nature of microarchitecture using electronic control and fluidic datapath.

In Section 4.1, we show experimental results to confirm these differences (all except the last which does not pertain to assays but to implementation and is discussed in the previous sections) and to validate the design decisions made on the basis of the implications of these differences. To that end, we show several assays and their hand-compiled equivalents using AIS. We pick the assays from several different domains including immunology, biochemistry, genomics, and synthesis (on-demand chemical synthesis).

In addition, we provide execution time breakdown of the assays in Section 4.2 to give an idea of how long these assays take, and confirm that AIS avoids the extra moves of dyadic instructions. We also briefly perform some instruction-level parallelism (ILP) analysis to show where execution time could be reduced and whether superscalar-like techniques could be useful. Finally, in Section 4.3 we give an idea of chip area and the area overhead incurred in going from an ASLoC to the AquaCore PLoC because PLoCs may include FFUs that are not needed by a specific assay but are needed for other assays.

## 4.1 Assays

In Figure 2 through Figure 6, we show the source and hand-compiled AIS assembly for five assays from different domains. These assays are scaled versions of their bench-scale counterparts and have been implemented in ASLoCs.

Figure 2(a) shows a polymerase chain reaction (PCR) which is a common operation in genomic assays that involve DNA [13]. PCR involves mixing a small quantity of DNA (as little as part of a single strand) with polymerase, followed by thermal cycling at different temperatures. Every mix and heat step copies (and hence doubles) the amount of DNA. The resultant amount is exponential in the number of thermal cycles. At the end, the assay uses a capillary electrophoresis (CE) step to separate and sense the amplified PCR products. CE separation is done by applying a large voltage across the two ends of the CE column filled with a buffer (a fluid that serves a medium for separation). Because different molecules have different charges, they flow through the buffer at different rates. An off-chip fluorescence sensor, placed at the end of the column, detects different fluorescence readings at different times, as different molecules reach the column end (i.e., separation and sensing occur in parallel). The off-chip sensor can be driven from AquaCore's dry part. Figure 2(b) shows the AIS assembly. In all our AIS code, we use *si* to denote the wet reservoir *i,* and *rj* to denote the dry register *j*. The PCR mixture is input from *port1* into reservoir *s1*, and the CE buffer from *port2* into *s2*. The *dry-move*

**PCR** **(a)**

```
1 Heat the PCR mixture to 95°C for 5 seconds
2 Heat the mixture to 53°C for 15 seconds
3 Heat the mixture to 72°C for 10 seconds
4 Repeat the thermal cycling 20 times
5 Send the mixture through the capillary
    electrophoresis (CE) column (5cm at 236V/cm)
6 Separate using separation medium for 180 s
7 Sense the fluorescence of the separated flow
```

**(b)**

```
Input port ip1  ;PCR mixture
Input port ip2  ;CE separation medium
RESULT[]        ;dry array for final results

PCR {
input s1, ip1
input s2, ip2

move heater1, s1                          ;5s

dry-mov r1, 20
dry-label loop:

incubate heater1, 95, 5                   ;6s
incubate heater1, 53, 15                  ;17s
incubate heater1, 72, 10                  ;12s

dry-dec r1
dry-bgt loop

move separator1.buf, s2                   ;5s
move separator1, heater1                  ;5s
separate.CE separator1, 236, 5, 180
sense.FL sensor1, RESULT                  ;180s
}

Total time = 895s
#reservoirs = 2
ASLoC area = unknown (length is 14.5mm)
```

**FIGURE 2: Polymerase Chain Reaction (PCR) (Genomics): (a) Source assay (b) AIS code**

instruction is a "dry" instruction to initialize the loop count in the dry register *r1* for the thermal cycling. The loop decrements *r1 (dry-dec)* and branches back (*dry-bgt*). *incubate* and *separate.CE* specify the incubation temperature and time, and separation length and electric field, respectively, as shown in Table 1. The heater contents are directly moved to the separator without going through a reservoir, illustrating the efficiency of AIS in fluid movement. The CE buffer is moved from *s2* into the CE column's buffer port identified by *separator1.buf*. The heated mixture in *heater1* is then sent into the CE column's regular input identified by *separator1*. *sense.FL* drives the off-chip sensor which stores its readings in the array *RESULT*. Note that all the operand volumes are implicit and are translated to an implementation-specific volume (e.g., 100 nl) by our run-time system. The numbers to the right of the instructions are latencies which we discuss in Section 4.2. This assay assumes that the input mixture is mixed off chip. The ingredients can be mixed on chip though doing so would use more reservoirs.

Figure 3(a) shows a biochemical immunoassay to sense different chemicals in neurotransmitters [6]. The assay uses a sample extracted from brain serum and mixes it with a reagent. The mixture is separated using electrophoresis as done in PCR (Figure 2). Figure 3(b) is a straight-forward compilation of the immunoassay.

Figure 4(a) shows a genomic assay to sense the different base pairs in a small DNA template [4]. The template first undergoes amplification (similar to PCR in Figure 2), followed by a purification step. Purification is achieved by passing the template through an affinity-based capture gel (where the capture gel holds on to the DNA and all other unwanted material is separated out). Finally, the purified template is sent through a CE column (with a separation gel) for separation and sensing. Figure 4(b) shows the AIS assembly. *mix*, *separate.CE*, and *sense.FL* instructions are similar to those of PCR. We use *separate.AF* for the affinity-based separation step. We dispose of the separator's flow-through which is identified by *separator1.flow-through,* and move the rest to the CE separator *separator2*.

**neurotransmitter-sensing** **(a)**

```
1 Mix the sample with reagent for 50s
2 Flow the mixture into a capillary
    electrophoresis (CE) column (9cm at 223V/cm)
3 Use an electrophoresis buffer as separation
    medium
4 Sense the fluorescence of the separated flow
```

**(b)**

```
Input port ip1  ;Sample
Input port ip2  ;Reagent(OPA)
Input port ip3  ;Elecrtrophoresis buffer
RESULT[]        ;dry array for final results

neurotransmitter-sensing {
input s1, ip1
input s2, ip2
input s3, ip3

move mixer1, s1                           ;5s
move mixer1, s2                           ;5s
mix mixer1, 50                            ;50s

move separator1.buf, s3                   ;5s
move separator1, mixer1                   ;5s
separate.CE separator1, 223, 9
sense.FL sensor1, RESULT                  ;22s
}

Total time = 92s
#reservoirs = 3
ASLoC area = 2.5cm x 1.5cm = 3.75cm^2 + CE column
```

**FIGURE 3: Separations-based sensing of neurotransmitters (immunoassay, biochemistry): (a) Source assay (b) AIS code**

Synthesizing chemicals at the nano and pico scales is more efficient in terms of yield percentage and safer (if the chemical is toxic) than at the macro scale. Figure 5(a) shows an example of a synthesis assay where fluoride-radiolabeled molecular imaging probe is synthesized through a series of mix and concentrate steps [14]. The AIS assembly is in Figure 5(b). One peculiarity is that the assay specifies mixing and concentrating simultaneously, which would require a separate mix-and-concentrate FFU. Instead, we first mix the reagents for the specified time and then concentrate the mixture for the same time using the first *concentrate.EV*. While this serialization is slower, it avoids a new FFU. If such mix and concentrate operations are common (our assays do not have other instances), then using a separate FFU would be justified.

A large set of biochemistry assays aim at detecting levels of certain molecules. Figure 6(a) shows an assay for detecting the concentration level of glucose. Given a standard glucose concentration, a calibration step is performed against different dilutions of this standard using a colorimetric (optical) sensor, and applying linear regression on the data points to get a best-fit line. Then the sample is sensed, and the reading is checked against the line to

**Sanger-DNA-Sequencing** **(a)**

Amplification:
1 Heat the DNA template (in primer) to 95°C for 12 seconds
2 Heat the DNA template to 60°C for 55 seconds
3 Repeat the above thermal cycling 35 times
4 Perform an affinity separation using capture gel
5 Dispose of the flowthrough
6 Separate the resultant using the separation gel in a capillary electrophoresis (CE) column for 34 min (30cm at 167V/cm)
8 Sense the fluorescence of the separated flow

**(b)**

```
Input port ip1  ;Capture gel
Input port ip2  ;Separation gel
Input port ip3  ;DNA template
RESULT[]        ;dry array for final results

Sanger-DNA-Sequencing {
input s1, ip1
input s2, ip2
input s3, ip3

move heater1, s3                          ;5s

dry-mov r1, 35
dry-label loop:

incubate heater1, 95, 12                  ;14s
incubate heater1, 60, 55                  ;57s

dry-dec r1
dry-jgt loop

move separator1.gel, s1                   ;5s
move separator1, heater1                  ;5s
separate.AF separator1, 1200              ;1200s
dispose separator1.flow-through           ;5s

move separator2.buf, s2                   ;5s
move separator2, separator1               ;5s
separate.CE separator2, 167, 30, 2040
sense.FL sensor1, RESULT                  ;2040s
}

Total time = 95.5 min = 3730s
#reservoirs = 3
ASLoC area (very sparse) = 40cm^2 (half circle of
100mm diameter)
```

**FIGURE 4: Sanger DNA sequencing (Genomics): (a) Source assay (b) AIS code**

obtain the sample's glucose concentration [22]. The AIS equivalent is shown in Figure 6(b). One difference from previous assays is that the *moves* use the optional relative-volume field (Table 1) to specify the relative volumes being mixed. The assay needs to mix 1, 2, 4, and 8 portions of the reagent with 1 portion of the glucose, and 1 portion of the reagent with 1 portion of the sample. Accordingly, a simple look-ahead calculation (Section 3.3) determines that the input volume of the reagent should be divided into 1 + 2 + 4 + 8 + 1 = 16 portions. Then the relative volumes of 1, 2, 4, and 8 are translated to 6, 12, 24, and 48 nl respectively, assuming the implementation-specific input volume is 100 nl (100/16 = 6.25 but we round it down to 6 for convenience). The only new instruction in this assay, *sense.OD*, is used for colorimetric (optical density) sensing done by an off-chip sensor like the fluorescence sensor discussed before. One interesting point is that the dry code to obtain the best-fit curve through the calibration data and the concentration of the sample from the curve is present inline with the wet code.

**Imaging-Probe-Synthesis** **(a)**

1 Mix fluoride ions with ion exchange beads and concentrate by evaporation at 100°C for 30s, 120°C for 30s, 135°C for 180s
2 Solvent exchange from water to acetonitrile by mixing with MeCN and heating at 100°C for 30s and 120°C for 50s
3 Solvent exchange from MeCN to water by mixing with HCl
4 Acidic hydrolysis of fluorinated intermediate by evaporation at 60°C for 1 min to obtain [$^{18}$F]FDG

**(b)**

```
Input port ip1  ;Ion exchange beads (in buffer)
Input port ip2  ;Fluoride ions F-
Input port ip3  ;MeCN solution
Input port ip4  ;HCl

Imaging-Probe-Synthesis {
input s1, ip1
input s2, ip2
input s3, ip3
input s4, ip4

move mixer1, s1                           ;5s
move mixer1, s2                           ;5s
mix mixer1, 30                            ;30s

move heater1, mixer1                      ;5s
concentrate.EV heater1, 100, 30           ;30s
concentrate.EV heater1, 120, 30           ;31s
concentrate.EV heater1, 135, 180          ;181s

move mixer1, s3                           ;5s
move mixer1, heater1                      ;5s
mix mixer1, 30                            ;30s

move heater1, mixer1                      ;5s
incubate heater1, 100, 30                 ;30s
incubate heater1, 120, 50                 ;51s

move mixer1, heater1                      ;5s
move mixer1, s4                           ;5s
mix mixer1, 60                            ;60s

move heater1, mixer1                      ;5s
concentrate.EV heater1, 60, 60            ;60s
}
Total time = 548s
#reservoirs = 4
ASLoC area = 13mm x 9mm = 117 mm^2
```

**FIGURE 5: Imaging Probe Synthesis: (a) Source assay (b) AIS code**

We now analyze these assays from the three aspects discussed at the beginning of Section 4, namely, coverage, the single-use property of intermediates, and the nature of operand volumes in fluidics (variable, implicit, and relative).

From the above assays, we see that AIS and AquaCore can easily cover these assays from diverse domains using relatively few opcodes. Of these opcodes, *mix* and *incubate* are common operations and hence, should be supported by any AquaCore variant. However, we see that assays differ in separate and sense operations. While the first three assays — PCR, immunoassay, and DNA sequencing — use electrophoresis as their separation method and fluorescence-based sensing, the synthesis assay uses no separation and sense steps whereas the glucose concentration assay uses colorimetric sensing. As we show in Section 4.3, because the CE column occupies large chip area, it may not be desirable to include a

**glucose-detection** **(a)**

```
1 Dilute the standard glucose concentration by
    mixing with reagent to get 1:1, 1:2, 1:4 and
    1:8 concentrations
2 Sense the different concentrations using a
    colorimetric sensor
3 Draw calibration curve for the different
    concentrations
4 Mix the sample with the reagent in a 1:1 ratio
5 Sense using colorimetric sensor
6 Obtain the concentration of the sample from
    the calibration curve
```

**(b)**

```
Input port ip1  ;Standard glucose
Input port ip2  ;Reagent
Input port ip3  ;Sample
RESULT[1..5]    ;dry array for final results

glucose-detection {
input s1, ip1
input s2, ip2
input s3, ip3

move mixer1, s1, 1                      ;5s
move mixer1, s2, 1                      ;5s
mix mixer1, 10                          ;10s
move sensor1, mixer1                    ;5s
sense.OD sensor1, RESULT[1]             ;30s

move mixer1, s1, 1                      ;5s
move mixer1, s2, 2                      ;5s
mix mixer1, 10                          ;10s
move sensor1, mixer1                    ;5s
sense.OD sensor1, RESULT[2]             ;30s

move mixer1, s1, 1                      ;5s
move mixer1, s2, 4                      ;5s
mix mixer1, 10                          ;10s
move sensor1, mixer1                    ;5s
sense.OD sensor1, RESULT[3]             ;30s

move mixer1, s1, 1                      ;5s
move mixer1, s2, 8                      ;5s
mix mixer1, 10                          ;10s
move sensor1, mixer1                    ;5s
sense.OD sensor1, RESULT[4]             ;30s

<dry routine to get best line fit for RESULT[1..4]>

move mixer1, s3, 1                      ;5s
move mixer1, s2, 1                      ;5s
mix mixer1, 10                          ;10s
move sensor1, mixer1                    ;5s
sense.OD sensor1, RESULT[5]             ;30s

<dry routine to get concentration from line
    given RESULT[5]>
}
Total time = 275s
#reservoirs = 3
ASLoC area = unknown
```

**FIGURE 6: Glucose detection (Biochemistry): (a) Source assay (b) AIS code**

CE column if the target domains do not typically use electrophoresis-based separation. Consequently, we envision a few AquaCore variants each of which targets domains with similar needs. For instance, AquaCore variants targeting genomics and immunoassays should include a CE column as these assays typically use electrophoresis-based separation. On the other hand, AquaCore variants targeting chemical synthesis and biochemistry assays may opt not to include a CE column.

We also see that each intermediate result is used only once in a later step, usually the subsequent step. In the next section, we show that moving fluids is slow and should be avoided. These results support our decision to use storage-less operands, avoiding unnecessary movement of fluids back and forth between the register file and FFUs, and instead move directly from the producer FFU to the consumer FFU.

Finally, we see that the operand volumes across these assays may vary and the same opcode may operate on different volumes in different assays or even in the same assay (e.g., the *mix* instructions in the glucose assay in Figure 6(b)). We also see that most of these assays use implicit volumes and do not specify any (relative or absolute) volumes. Our approach of using implicit operand volumes matches this practice. Only the glucose assay specifies relative volumes. AIS avoids forcing the assay writer to use absolute volumes in this assay, allowing the AIS code to be portable across same-technology implementations.

## 4.2 Execution Time Analysis

High-level estimates for instructional latencies are shown to the right of the instructions in Figure 2(b) through Figure 6(b). These estimates are obtained from the corresponding ASLoC chips, as we explain below. The total execution times are calculated assuming completely sequential execution.

PCR (Figure 2(b)) takes 895 s (shown at the bottom) of which the first move takes 5s and the heat instructions in each iteration of the loop take 6+17+12 = 35 s and the loop takes 690 s. *move*'s latency is estimated by using a typical peristaltic flow rate of 2 mm/s [16], and by conservatively assuming every move traverses half of one dimension of the chip and using a dimension of 2 cm (discussed in the next section). The latency of heat instructions have two components: (1) time to reach the specified temperature, and (2) incubation time as specified by the assay. For example the *third* heat takes 12 s of which 1.5 s is spent in reaching the specified temperature (72 $^{o}$C) from the previous temperature (53 $^{o}$C), and 10 s is the incubation time. Time to reach a specified temperature from an initial temperature is mostly a function of the temperature difference, and is largely independent of the reagent being heated (or cooled) because of the small amount of the reagent. We assume a heating rate of 20 $^{o}$C/s [13]. *separate.CE* and *sense.FL* (occur in parallel, as discussed in Section 4.1) take the remaining 180 s based on the specified separation length and voltage [13].

In the immunoassay (Figure 3(b)), *mix* takes the specified 50 s and *separate.CE and sense.FL* take 22 s based on the specified separation length and voltage, for a total of 92s. The mix time is based on a passive, flow-based Y mixer from [6] whereas we use an active mixer (Section 2) which would be faster but we conservatively assume the slower time.

The DNA assay (Figure 4(b)) takes 3730 s of which the amplification (heat-cycling) loop takes (14 + 57) * 35 = 2485 s. *separate.AF* takes the specified 20 minutes, and *separate.CE* and *sense.FL* take 34 minutes, based on the specified separation length and voltage.

The imaging probe synthesis (Figure 5(b)) takes a total of 548 s which are spent in a series of *mix* and *heat* instructions, whose latencies have been explained before.

Finally, the glucose assay (Figure 6(b)) takes 275 s (ignoring the dry code which is orders of magnitude faster than wet instructions) of which the calibration step consisting of four sets of *mix*,

*move*, and *sense.CL* takes 220 s (= 4*(5+5+10+5+30)). The detection step for the actual sample is another set of *mix, move* and *sense.CL* taking another 55 s.

These numbers provide a high-level estimate of execution times in fluidics. Though the assays are short they run for long durations compared to computer programs of similar lengths.

Also, observe that if the instructions were dyadic, then each *incubate* instruction in the PCR loop (Figure 2(b)) would have added as much as 10 s to move the reagent between the reservoirs and the heater. In contrast, we move the reagent from the reservoir to the heater only once outside the loop. Thus, dyadic instructions would have incurred 6 transfers per loop iteration (two per *incubate*) for a total of 122 transfers 120 moves in the loop and two moves required for separation), compared to the 3 transfers in the AIS code. Because each transfer involves many actuations of the peristaltic pumps (Section 2), reducing the number of transfers improves reliability. Also, the run time would have increased by 30 s (10 s for each incubate) * 20 (loop iterations) - 5 s (time for our move) = 595 s, which is a speedup of 66%. In the immunoassay, DNA, imaging probe synthesis, and glucose assays, dyadic instructions would have incurred a total of 5, 145, 21, and 20 transfers, respectively, as compared to 4, 6, 9, and 15 transfers in the AIS code. The corresponding speedups are 5%, 19%, 10%, and 9%, respectively.

Analyzing the ILP in the assays, we see that some of the assays are too short or are single long dependence chains with a little ILP in moves (e.g., the first four assays - PCR, immunoassay, DNA, and imaging probe synthesis). The three channels in our microarchitecture could exploit this move parallelism (in addition to improving reliability, as discussed in Section 3.4). The glucose assay, which is the last, has higher ILP in the calibration step where the different concentrations and the sample could be mixed and sensed in a parallel or pipelined manner. Moreover, sequences of *incubate* or *concentrate* instructions (e.g., PCR and imaging probe synthesis) can also be sped up by using multiple heaters so that the time to adjust the next heater to the required temperature can be overlapped with the previous incubate (but in our examples the time to heat is much smaller than the incubation period to be worth overlapping by using another heater). Also, recall that the PCR could be written to mix the ingredients on chip which could increase PCR's ILP.

## 4.3 Chip Area Analysis

We compare our microarchitecture's chip area versus the chip areas of the ASLoCs for each of the assay. Typical estimates of individual FFU dimensions are given in Table 2. We scale down the area of the valves from 1 $mm^2$ [11] to 0.1 $mm^2$ to match the size of the rest of the components. We assume 10 reservoirs and 3 channels for flexibility, one mixer, one heater, one of each of the two separators, and 4 input and output ports each. Each reservoir, input port, output port, and FFU port requires 3 valves (one for each channel). Recall from Section 2 that each peristaltic channel requires three inter-locking valves. Assuming 3 FFU ports per FFU, we need about 99 latching valves. Recall from Section 3.4 that the valves are controlled using a demultiplexer. Grover et al. [11] construct a 1-1024 demultiplexer containing about $2^{11} = 2048$ half-valves (equivalent to one P or N transistor) each of which is 1 $mm^2$ for a total of about 40 $cm^2$. Because we assume a scaled-down valve of 0.1 $mm^2$, and because we need only a 1-128 demultiplexer this area will reduce to 0.5 $cm^2$. The components in the table take up about 110 $mm^2$, the latching valves occupy 99 * 0.1 ~= 10 $mm^2$, the channels occupy 3 * 20 mm length * 0.1 mm width = 6 $mm^2$, and the multiplexer is 0.5 $cm^2$ for a total chip area of 1.76 $cm^2$. Additionally, the chip uses a total of 8 off-chip solenoid pumps to control the valves via the demultiplexer. Note the CE column's large footprint suggesting the AquaCore variants discussed in Section 4.1. The channels span the components and the valves (about 120 $mm^2$ area), but not the CE column or multiplexer which are off to the sides of the chip. Consequently, our estimate of channel lengths of 2 cm is reasonable.

**Table 2: Dimensions of FFUs.**

(** modified the depth to match our etching depth*)

| **Components** | Dimensions l x w (d = 100 µm) | Area |
|---|---|---|
| **Valve (scaled down from [11])** | 100µm x 100µm | 0.1 $mm^2$ |
| **Reservoir (100 nl volume)** | 1mm x 1mm | 1$mm^2$ |
| **Mixer (pressurized using PDMS) [16]** | 1mm x 1mm | 1 $mm^2$ |
| **Heater [16]** | 1mm x 1mm | 1$mm^2$ |
| **Separator (AF) [4]*** | 1.45mm x 800µm | 1.16 $mm^2$ |
| **Separator (CE) [4, 13, 6]*** | 8 of (2cm x 600µm) | 96 $mm^2$ |
| **Sensors** | off-chip | 0 $mm^2$ |

The total chip area should be compared to those of the immunoassay ASLoC at 3.75 $cm^2$ and the imaging probe ASLoC at 1.17 $cm^2$ (the DNA ASLoC layout is too sparse to make any comparisons). However, there are four caveats: (1) our estimates are high-level, (2) we assume that the valves can be scaled down from 1 $mm^2$ to 0.1 $mm^2$, (3) our estimates assume tight packing without accounting for spacing and routing constraints, and (4) while the ASLoCs are real chips, our AquaCore microarchitecture is a paper design. With these caveats in mind, our results indicate that the area overhead of programmability may be reasonable.

## 4.4 Discussion

We acknowledge that compared to computer benchmarks, our assays are closer to kernels than full-scale applications. Other assays we have examined are also of similar size and complexity. This trend may be due to the non-automated nature of bench-scale experiments and the non-programmable nature of ASLoCs, which discourage composing these assays into larger applications. If the PLoC approach succeeds, then larger applications may come into existence and may change some of the above observations about assay behavior, warranting change to the AquaCore microarchitecture. We not only recognize but also anticipate this possibility, and point out that modifying the architecture to fit the changing workload fuels advancements in all architectures — computer or fluidic.

We note that because the channels in the microarchitecture are involved in *all* fluid transfers, the valves forming the peristaltic pumps may incur a disproportionate number of actuations and may

degrade reliability. Consequently, alternative pumping methods (e.g., electro-kinetic and pneumatic) may have to replace peristalsis in the channels.

# 5 CONCLUSIONS AND FUTURE WORK

Lab-on-a-chip (LoC) technology enables miniaturization and integration of biological and chemical analyses to a single chip. The technology has the potential to achieve faster, cheaper, and higher-precision analyses compared to conventional bench-scale methods in diverse domains such as proteomics, genomics, biochemistry, virology, cell biology, and chemical synthesis. However, to date, LoCs have been designed as application-specific chips which incurs significant design effort, turn-around time, and cost, and degrades designer and user productivity. We advocate a programmable LoC (PLoC) to address these limitations.

We proposed a comprehensive fluidic instruction set, called AquaCore Instruction Set (AIS), and a fluidic microarchitecture, called AquaCore, to implement AIS. AIS and AquaCore differ from their computer counterparts in four key aspects, namely, coverage, number of operands in instructions, nature of operand size, and heterogeneity of the microarchitecture. We demonstrated the use of AIS and AquaCore by hand-compiling real-world microfluidic assays from genomics, immunology, chemical synthesis, and biochemistry. Our initial high-level estimates indicate that the area overhead of programmability may be acceptable.

We view AIS and AquaCore as a first step in the design of instruction sets and microarchitectures for programmable microfluidics. Much work remains to be done. We need to find architectural techniques to enhance the reliability of AquaCore. More assays, including those from domains not covered in this paper, need to be ported to AIS to confirm the generality of the PLoC approach. We also need to create a compilation toolchain to deliver the real promise of PLoCs. Finally, we need to develop prototypes to show proof-of-concept of the PLoC approach.

# REFERENCES


[1] Agilent Lab-on-a-chip Products, http://www.chem.agilent.com/Scripts/PCol.asp?lPage=50.

[2] Caliper Life Sciences, LabChip Products, http://www.caliperls.com/technology_partners/microfluidic/.

[3] P. Auroux et al. Micro total analysis systems. 2. Analytical standard operations and applications. *Analytical Chemistry*, 74(12):2637–2652, 2002.

[4] R. G. Blazej, P. Kumaresan, and R. A. Mathies. Microfabricated bioprocessor for integrated nanoliter-scale Sanger DNA sequencing. *Proceedings of the National Academy of Sciences (PNAS)*, 103(19):7240–7245, 2006.

[5] J. Butts and G. Sohi. Characterizing and predicting value degree of use. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 15–26, 2002.

[6] N. A. Cellar and R. T. Kennedy. A capillary-pdms hybrid chip for separations-based sensing of neurotransmitters in vivo. *Lab on a Chip*, 6(9):1205–1212, June 2006.

[7] E. Clark, J. Fanguy, and C. Henry. High-throughput multi-analyte screening for renal disease using capillary electrophoresis. *Pharmaceutical and Biomedical Analysis*, 25(5-6):795–801, 2001.

[8] K. W. Current et al. A high-voltage integrated circuit engine for a dielectrophoresis-based programmable micro-fluidic processor. In *Proceedings of the International Conference on MEMS, NANO and Smart Systems*, pages 153–158, 2005.

[9] P. Dittrich, K. Tachikawa, and A. Manz. Micro total analysis systems. Latest advancements and trends. *Analytical Chemistry*, 78(12):3887–908, 2006.

[10] C. Goll et al. Microvalves with bistable buckled polymer diaphragms. *Micromechanics and Microengineering*, 6(1):77–79, 1996.

[11] W. Grover et al. Development and multiplexed control of latching pneumatic valves using microfluidic logical structures. *Lab on a Chip*, 6:623–631, 2006.

[12] E. Hasselbrink, T. Shepodd, and J. Rehm. High-pressure microfluidic control in lab-on-a-chip devices using mobile polymer monoliths. *Analytical Chemistry*, 74(19):4913–4918, 2002.

[13] E. T. Lagally, C. A. Emrich, and R. A. Mathies. Fully integrated pcr-capillary electrophoresis microsystem for dna analysis. *Lab on a Chip*, 1(2):102–107, Oct. 2001.

[14] C.-C. Lee et al. Multistep Synthesis of a Radiolabeled Imaging Probe Using Integrated Microfluidics. *Science*, 310(5755):1793–1796, 2005.

[15] K. Mogensen, H. Klank, and J. Kutter. Recent developments in detection for microfluidic systems. *Electrophoresis*, 25(21-22):3498–512, 2004.

[16] N. Nguyen and S. Wereley. *Fundamentals and Applications of Microfluidics*. Artech House, 2002.

[17] K. Oh et al. World-to-chip microfluidic interface with built-in valves for multichamber chip-based PCR assays. *Lab on a Chip*, 5:845–850, 2005.

[18] J. Pan, D. VerLee, and M. Mehregany. Latched valve manifolds for efficient control of pneumatically actuated valve arrays. In *Proceedings of the International Conference on Solid State Sensors and Actuators*, volume 2, pages 817–820, 1997.

[19] M. Pollack, A. Shenderov, and R. Fair. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab on a Chip*, 2(1):96–101, 2002.

[20] D. Reyes et al. Micro total analysis systems. 1. Introduction, theory, and technology. *Analytical Chemistry*, 74(12):2623–2636, 2002.

[21] K. Shaikh et al. A modular microfluidic architecture for integrated biochemical analysis. *Proceedings of the National Academy of Sciences (PNAS)*, 102(28):9745–9750, 2005.

[22] V. Srinivasan et al. A digital microfluidic biosensor for multianalyte detection. In *Proceedings of the 16th Annual IEEE International Conference on Micro Electro Mechanical Systems*, pages 327–330, 2003.

[23] F. Su, K. Chakrabarty, and R. Fair. Microfluidics-Based Biochips: Technology Issues, Implementation Platforms, and Design-Automation Challenges. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(2):211–223, 2006.

[24] W. Thies et al. Abstraction layers for scalable microfluidic biocomputers. In *International Meeting on DNA Computing*, Seoul, Korea, June 2006.

[25] M. Unger et al. Monolithic Microfabricated Valves and Pumps by Multilayer Soft Lithography. *Science*, 288(5463):113–116, 2000.

[26] J. P. Urbanski et al. Digital microfluidics using soft lithography. *Lab on a Chip*, 6(1):96–104, Jan 2006.

[27] T. Vilkner, D. Janasek, and A. Manz. Micro total analysis systems. Recent developments. *Analytical Chemistry*, 76(12):3373–85, 2004.